Università degli Studi di Salerno

Anno Accademico 2019/2020

Corso di Ingegneria del Software

Object Design Document

V1.4

STAY SOFTWARE

Top Manager:

Prof. De Lucia Andrea

Team di sviluppo:

Nome e Cognome	Matricola
Aniello Mancusi	0512102610
Vincenzo Zito	0512100507

Revision History:

Autore	Data	Descrizione	Versione
Vincenzo Zito	*08/05/19*	*Struttura documento*	*v 1.0*
Aniello Mancusi	*09/05/19*	*Stesura introduzione*	*v 1.1*
Aniello Mancusi	*10/05/19*	*Stesura capitolo package*	*v 1.2*
Vincenzo Zito	*14/05/19*	*Revisione capitolo package*	*v 1.3*
Vincenzo Zito	*10/11/2020*	*Revisione documento*	*v1.4*

Sommario

Introduzione

Lo scopo dell'Object Design Document è quello di specificare i servizi che ogni sottosistema, che è stato presentato nel System Design Document, offre in termini di classi includendo operazioni, tipi, argomenti e signature in modo da avere una specifica completa. Questo documento serve come base per l'implementazione del progetto software.

Object Design Trade-offs

Interfaccia vs. Usabilità

L'interfaccia, grazie all'utilizzo di un'impostazione semplice e intuitiva, permette un uso facile (*Easy-Use*) della gestione del sistema e del relativo database di prodotti, da parte di chi amministra il negozio, e della navigazione all'interno del negozio per la ricerca e il relativo acquisto di prodotti, da parte dei clienti.

Tempo di rilascio vs Tolleranza ai fault

Una volta creata la struttura principale verrà dedicato tempo in più per la creazione di controllori in grado di gestire ogni eventuale errore della web application e verrà assicurato un minimo di 4 backup giornalieri che permetteranno di avere un'istantanea del database che, in caso di guasti, sarà subito disponibile per la rimessa in funzione dell'intera applicazione.

Sicurezza vs. Efficienza

Verrà implementato un modulo dedicato interamente all'autenticazione degli utenti solo dopo effettuata l'autenticazione, l'acquisto di beni/servizi (da parte dei clienti) e la modifica del database (da parte degli amministratori).

Comprensibilità vs. Tempo

La stesura del codice sarà suddivisa in parti e commentata per rendere il più leggibile possibile lo stesso e permettere a terzi eventuali modifiche strutturali.

Costi vs. Mantenimento

Grazie a un uso di materiale open source e l'utilizzo di commenti il codice sarà facilmente modificabile (implementazione di nuove funzioni o correzioni di errori) con costi contenuti. L'architettura three-tier richiederà tuttavia dei costi di gestione un po' più alti.

Interfaccia vs. Tempo di risposta

Il tempo di risposta tra server e interfaccia sono più che sufficienti a soddisfare le esigenze dei vari clienti che si collegheranno al Sistema per ricercare e/o acquistare beni/servizi. Ovviamente maggiore sarà la grandezza del database e maggiore sarà il tempo di risposta e ricerca nel database che ovviamente sarà indicizzato per abbassare ulteriormente questi tempi.

Linee Guida per la Documentazione delle Interfacce

Gli sviluppatori dovranno seguire alcune linee guida per la scrittura del codice:

Naming Convention

E' buona norma utilizzare nomi:

- Descrittivi
- Pronunciabili
- Di uso comune
- Lunghezza medio-corta
- Non abbreviati
- Evitando la notazione ungherese
- Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

Variabili

I nomi delle variabili devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Quest'ultime devono essere dichiarate ad inizio blocco, solamente una per riga e devono essere tutte allineate per facilitare la leggibilità.

Esempio: comuneNascita

E' inoltre possibile, in alcuni casi, utilizzare il carattere underscore "_", ad esempio quando utilizziamo delle variabili costanti oppure quando vengono utlizzate delle proprietà statiche.

Esempio:

Metodi

I nomi dei metodi devono cominciare con una lettera minuscola, e le parole seguenti con la lettera maiuscola. Il nome del metodo tipicamente consiste di un verbo che identifica una azione, seguito dal nome di un oggetto. I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile(). Le variabili dei metodi devono essere dichiarate appena prima del loro utilizzo e devono servire per un solo scopo, per facilitare la leggibilità. Esistono però casi particolari come ad esempio nell'implementazione dei model, dove viene utilizzata l'interfaccia CRUD.

Esempio: getMail(), setMail()

Classi e pagine

I nomi delle classi e delle pagine devono cominciare con una lettera maiuscola, e anche le parole seguenti all'interno del nome devono cominciare con una lettera maiuscola. I nomi di quest'ultime devono fornire informazioni sul loro scopo.

Esempio: User.php

Ogni file sorgente php contiene una singola classe e dev'essere strutturato in un determinato modo:

- Una breve introduzione alla classe
  L'introduzione indica: l'autore, la versione e la data.

```
/**
* @author nome dell'autore
* @version numero di versione della classe
* @since data d'implementazione
*/
```

- L'istruzione include che permette di importare all'interno della classe gli altri oggetti che la classe utilizza.

- La dichiarazione di classe caratterizzata da:
  1. Dichiarazione della classe pubblica
  2. Dichiarazioni di costanti

3. Dichiarazioni di variabili di classe
4. Dichiarazioni di variabili d'istanza
5. Costruttore
6. Commento e dichiarazione metodi.

Esempio:

```
class User
{
   private $codUtente;
   private $nome;
   private $cognome;
   private $codFiscale;
   private $dataNascita;
   private $provinciaNascita;
   private $comuneNascita;
   private $nazioneNascita;
   private $via;
   private $provinciaResidenza;
   private $comuneResidenza;
   private $cap;
   private $sesso;
   private $telefono1;
   private $telefono2;
   private $mail;
   private $password;

   public function __construct($codUt, $name, $surname, $codFi, $dataN, $provinciaN,
$comuneN, $nazioneN, $street, $provinciaR, $comuneR, $postalCode, $gender, $phone1,
$phone2, $email,
                        $pass)
   {
      $this->codUtente = $codUt;
```

```
    $this->nome = $name;
    $this->cognome = $surname;
    $this->codFiscale = $codFi;
    $this->dataNascita = $dataN;
    $this->provinciaNascita = $provinciaN;
    $this->comuneNascita = $comuneN;
    $this->nazioneNascita = $nazioneN;
    $this->via = $street;
    $this->provinciaResidenza = $provinciaR;
    $this->comuneResidenza = $comuneR;
    $this->cap = $postalCode;
    $this->sesso = $gender;
    $this->telefono1 = $phone1;
    $this->telefono2 = $phone2;
    $this->mail = $email;
    $this->password = $pass;
}

/**
 * @return mixed
 */
public function getCodUtente()
{
    return $this->codUtente;
}

/**
 * @param mixed $codUtente
 */
public function setCodUtente($codUtente)
{
    $this->codUtente = $codUtente;
```

```
}

/**
 * @return mixed
 */
public function getNome()
{
    return $this->nome;
}

/**
 * @param mixed $nome
 */
public function setNome($nome)
{
    $this->nome = $nome;
}

/**
 * @return mixed
 */
public function getCognome()
{
    return $this->cognome;
}

/**
 * @param mixed $cognome
 */
public function setCognome($cognome)
{
    $this->cognome = $cognome;
```

```
}

/**
 * @return mixed
 */
public function getCodFiscale()
{
    return $this->codFiscale;
}

/**
 * @param mixed $codFiscale
 */
public function setCodFiscale($codFiscale)
{
    $this->codFiscale = $codFiscale;
}

/**
 * @return mixed
 */
public function getDataNascita()
{
    return $this->dataNascita;
}

/**
 * @param mixed $dataNascita
 */
public function setDataNascita($dataNascita)
{
    $this->dataNascita = $dataNascita;
```

```
}

/**
 * @return mixed
 */
public function getProvinciaNascita()
{
    return $this->provinciaNascita;
}

/**
 * @param mixed $provinciaNascita
 */
public function setProvinciaNascita($provinciaNascita)
{
    $this->provinciaNascita = $provinciaNascita;
}

/**
 * @return mixed
 */
public function getComuneNascita()
{
    return $this->comuneNascita;
}

/**
 * @param mixed $comuneNascita
 */
public function setComuneNascita($comuneNascita)
{
    $this->comuneNascita = $comuneNascita;
```

```
}

/**
 * @return mixed
 */
public function getNazioneNascita()
{
    return $this->nazioneNascita;
}

/**
 * @param mixed $nazioneNascita
 */
public function setNazioneNascita($nazioneNascita)
{
    $this->nazioneNascita = $nazioneNascita;
}

/**
 * @return mixed
 */
public function getVia()
{
    return $this->via;
}

/**
 * @param mixed $via
 */
public function setVia($via)
{
    $this->via = $via;
```

```
}

/**
 * @return mixed
 */
public function getProvinciaResidenza()
{
    return $this->provinciaResidenza;
}

/**
 * @param mixed $provinciaResidenza
 */
public function setProvinciaResidenza($provinciaResidenza)
{
    $this->provinciaResidenza = $provinciaResidenza;
}

/**
 * @return mixed
 */
public function getComuneResidenza()
{
    return $this->comuneResidenza;
}

/**
 * @param mixed $comuneResidenza
 */
public function setComuneResidenza($comuneResidenza)
{
    $this->comuneResidenza = $comuneResidenza;
```

```
}

/**
 * @return mixed
 */
public function getCap()
{
    return $this->cap;
}

/**
 * @param mixed $cap
 */
public function setCap($cap)
{
    $this->cap = $cap;
}

/**
 * @return mixed
 */
public function getSesso()
{
    return $this->sesso;
}

/**
 * @param mixed $sesso
 */
public function setSesso($sesso)
{
    $this->sesso = $sesso;
```

```
}

/**
 * @return mixed
 */
public function getTelefono1()
{
    return $this->telefono1;
}

/**
 * @param mixed $telefono1
 */
public function setTelefono1($telefono1)
{
    $this->telefono1 = $telefono1;
}

/**
 * @return mixed
 */
public function getTelefono2()
{
    return $this->telefono2;
}

/**
 * @param mixed $telefono2
 */
public function setTelefono2($telefono2)
{
    $this->telefono2 = $telefono2;
```

```
}

/**
 * @return mixed
 */
public function getMail()
{
    return $this->mail;
}

/**
 * @param mixed $mail
 */
public function setMail($mail)
{
    $this->mail = $mail;
}

/**
 * @return mixed
 */

public function getPassword()
{
    return $this->password;
}

/**
 * @param mixed $password
 */
public function setPassword($password)
{
```

```
        $this->password = $password;
    }
}
```

Definizioni, acronimi e abbreviazioni

Acronimi

RAD: Requirements Analysis Document

SDD: System Design Document

ODD: Object Design Document

CRUD: Create Read Update Delete

Abbreviazioni

DB: DataBase

Riferimenti

Documento SDD del progetto StaySoftware

Documento RAD del progetto StaySoftware

Package

Il software ha un'architettura three-tier e i livelli che si distinguono sono i seguenti:

1) **Interface Layer:** livello che gestisce la parte grafica del sistema software
2) **Application Logic Layer:** gestisce la parte logica e le relative query che vengono sottomesse allo storage layer.
3) **Storage Layer:** questo livello gestisce l'archiviazione persistente dei dati.

I layer sono suddivisi in package ed ognuno di questi contiene degli oggetti che andranno ad implementare le varie funzionalità del sistema che andremo a sviluppare. I package vengono utilizzati per riunire le classi, logicamente correlate, che forniscono dei servizi simili.

Descrizione dei layer

Interface Layer

Modulo	Descrizione
Interfaccia web	Questo modulo descrive l'interfaccia grafica con cui l'utente interagisce e che interpreta i form sottomessi e quindi di ricevere e interpretare gli input.

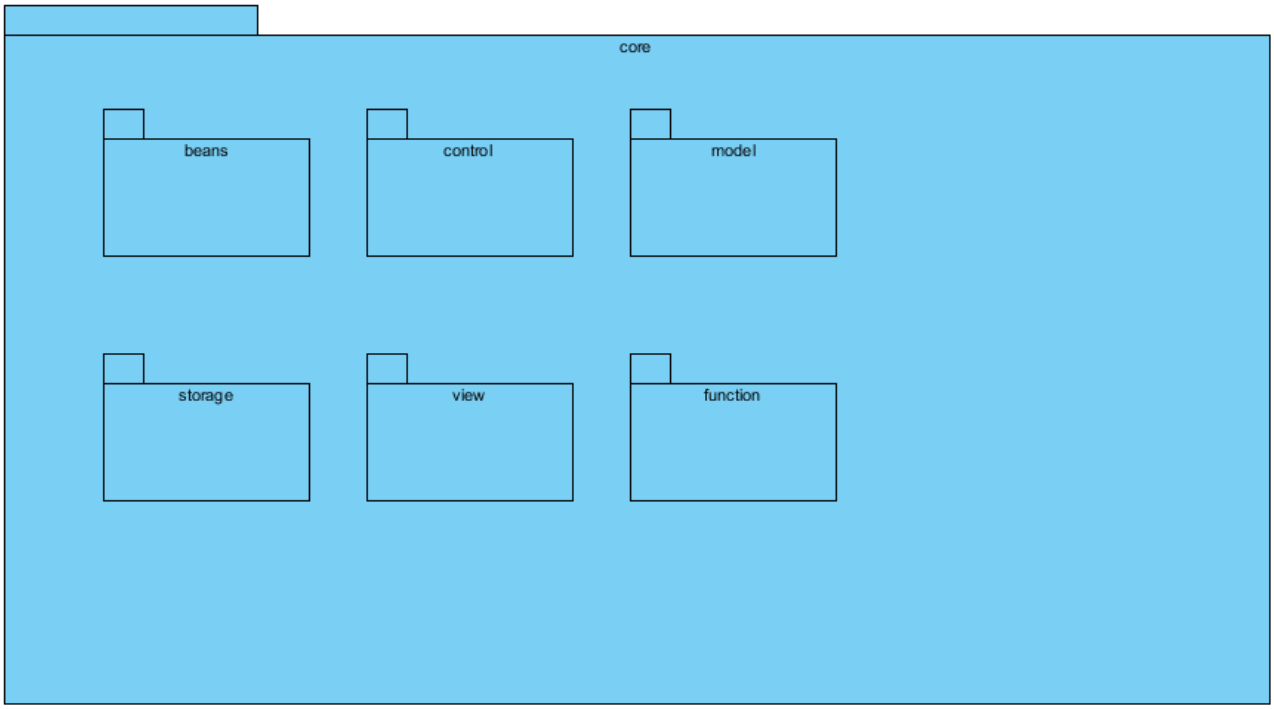
Application Logic Layer

Modulo	Descrizione
Beans	Package contenente le classi utilizzate per rappresentare gli oggetti entity.
Control	Package contenente le classi utilizzate per validare i dati inseriti.
Model	Package contenente le classi utilizzate per le operazione di gestione utente, gestione ordine, gestione acquisto, gestione prodotti, gestione riparazione e gestione admin
Storage	Il modulo si occupa della comunicazione con il database

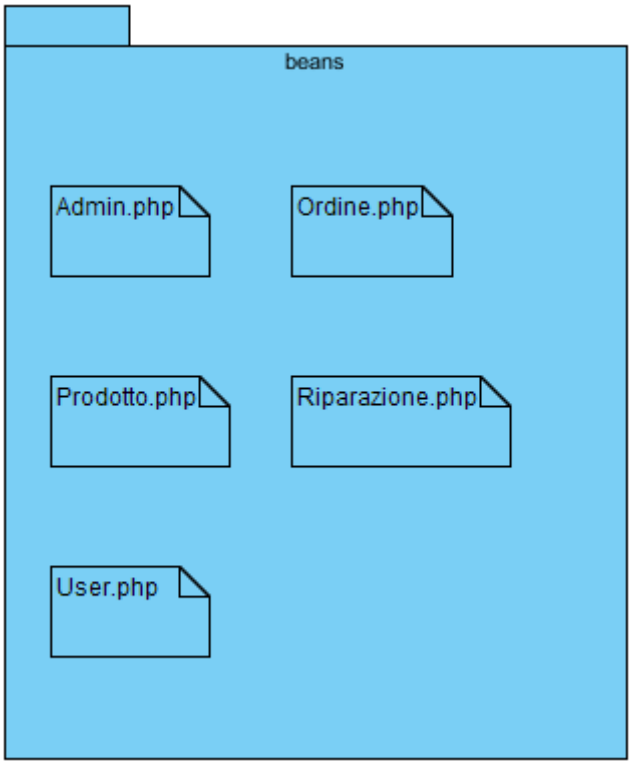
Storage Layer

Modulo	Descrizione
Database	Gestisce le richieste di dati in entrata e in uscita. Si occupa del database e della sua gestione.

Packages core

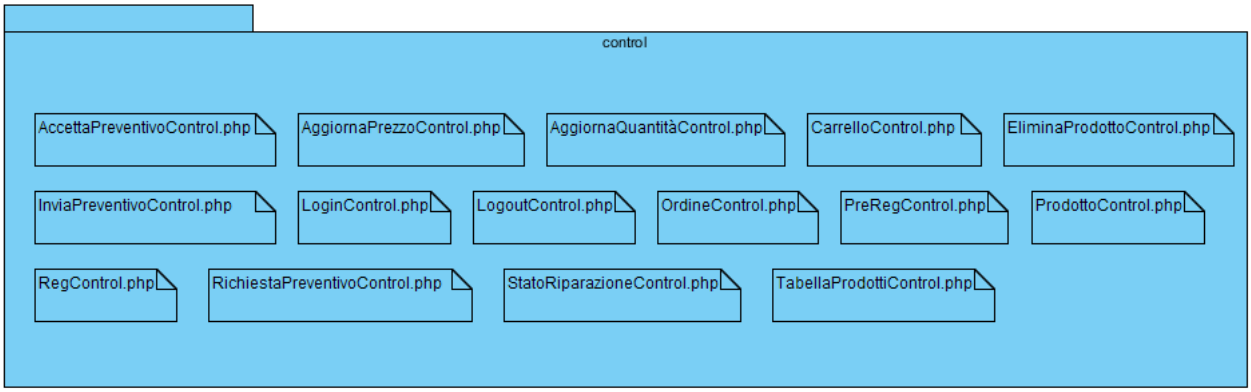


Package Beans



Classe:	Descrizione:
User.php	Classe utilizzata per rappresentare l'entità User
Admin.php	Classe utilizzata per rappresentare l'entità Admin
Ordine.php	Classe utilizzata per rappresentare l'entità Ordine
Prodotto.php	Classe utilizzata per rappresentare l'entità Prodotto
Riparazione.php	Classe utilizzata per rappresentare l'entità Riparazione

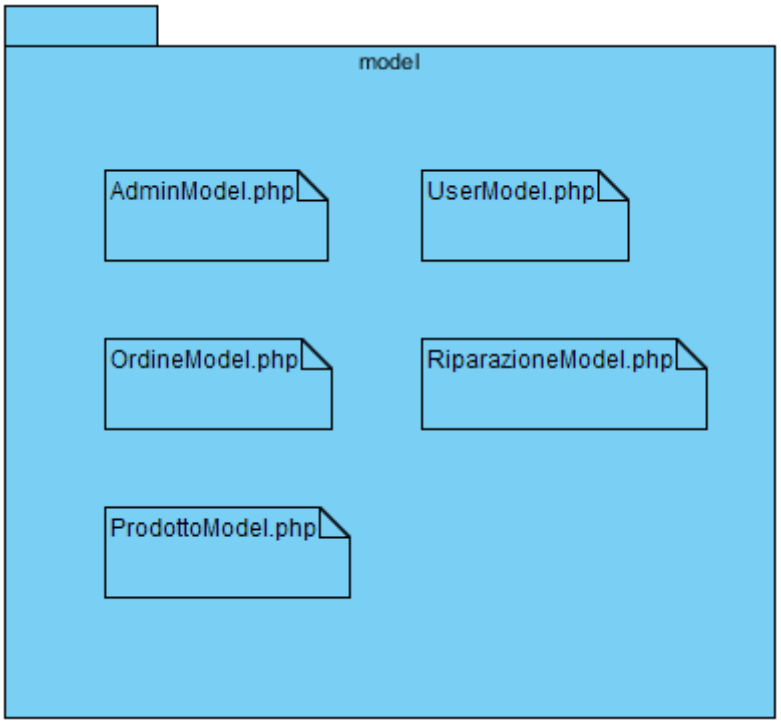
Package Control



Classe:	Descrizione:
LoginControl.php	Si occupa della fase di autenticazione dell'utente.
LogoutControl.php	Si occupa della fase di de-autenticazione dell'utente.
PreRegControl.php	Si occupa di verificare se l'utente è già registrato
RegControl.php	Si occupa della registrazione dell'utente e di verificare i dati inseriti
AccettaPreventivoControl.php	Si occupa dell'accettazione del preventivo.
AggiornaPrezzoControl.php	Si occupa di aggiornare il prezzo dei prodotti

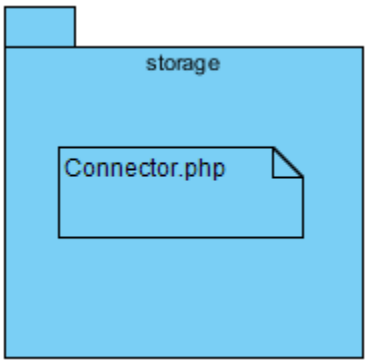
AggiornaQuantitaControl.php	Si occupa di aggiornare la quantità dei prodotti
CarrelloControl.php	Si occupa della gestione del carrello
EliminaProdottoControl.php	Si occupa di eliminare un prodotto
InviaPreventivoControl.php	Si occupa di inviare un preventivo
OrdineControl.php	Si occupa della gestione dell'ordine
ProdottoControl.php	Si occupa della gestione dei prodotti
RichiestaPreventivoControl.php	Si occupa di richiedere un preventivo.
StatoRiparazioneControl.php	Si occupa di gestire lo stato di una riparazione
TabellaProdottiControl.php	Si occupa di gestire la tabella prodotti

Package Model



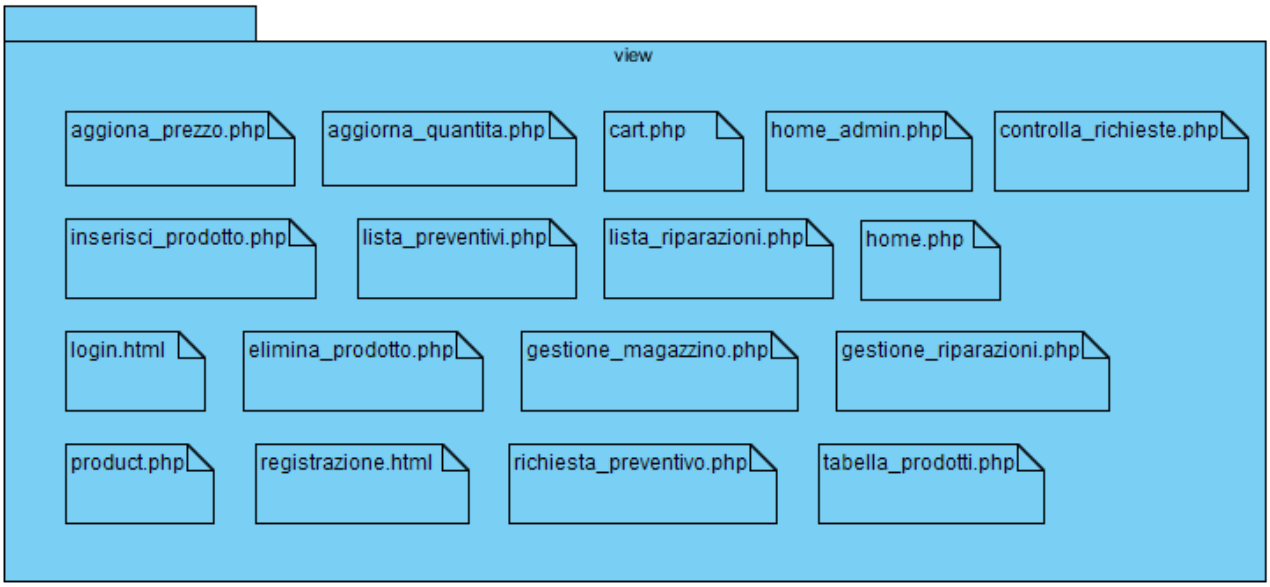
Classe:	Descrizione:
UserModel.php	Permette di manipolare tutte le informazioni relative all'entità utente.
AdminModel.php	Permette di manipolare tutte le informazioni relative all'entità admin.
ProdottoModel.php	Permette ad un admin di inserire, modificare, cancellare, ricercare e visualizzare un prodotto.
RiparazioneModel.php	Permette di manipolare tutte le informazioni relative all'entità riparazione.
OrdineModel.php	Permette di manipolare tutte le informazioni relative all'entità ordine.

Package Storage



Classe:	Descrizione:
Connector.php	Si occupa della connessione con il database.

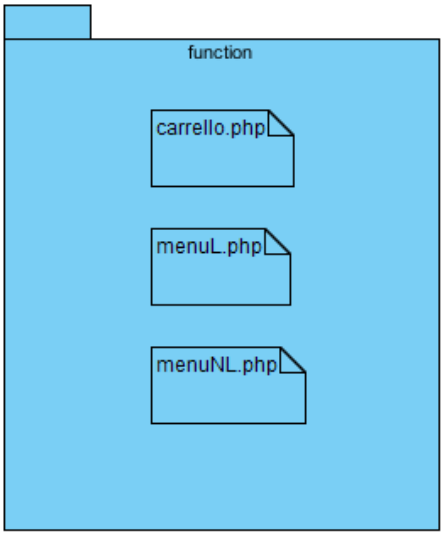
Package View



Classe:	**Descrizione:**
aggiorna_prezzo.php	Permette la visualizzazione della pagina per aggiornare il prezzo di un prodotto
aggiorna_quantita.php	Permette la visualizzazione di una pagina per aggiornare la quantità di un prodotto
cart.php	Permette la visualizzazione della pagina dedicata al carrello
home_admin.php	Permette la visualizzazione della pagina dedicata alla home per l'admin
controlla_richieste.php	Permette al visualizzazione della pagina per controllare le richieste preventivo
inserisci_prodotto.php	Permette la visualizzazione della pagina dedicata per l'inserimento di un prodotto
lista_preventivi.php	Permette la visualizzazione della pagina dedicata per controllare la lista preventivi
lista_riparazioni.php	Permette la visualizzazione della pagina dedicata per controllare la lista riparazione
home.php	Permette la visualizzazione della pagina dedicata alla home del sito una volta loggato
login.html	Permette la visualizzazione della pagina dedicata al login
elimina_profotto.php	Permette di visualizzare la pagina dedicata all'eliminazione di un prodotto
gestione_magazzino.php	Permette la visualizzazione della pagina dedicata alla gestione del magazzino da parte dell'admin
gestione_riparazioni.php	Permette la visualizzazione della pagina dedicata alla gestione delle riparazioni da parte dell'admin

product.php	Permette la visualizzazione della pagina dedicata al singolo prodotto
registrazione.html	Permette la visualizzazione della pagina dedicata alla registrazione
richiesta_preventivo.php	Permette la visualizzazione della pagina dedicata alla sottomissione di una richiesta di preventivo
tabella_prodotti.php	Permette la visualizzazione della pagina dedicata alla tabella prodotti

Package Function



Classe:	**Descrizione:**
carrello.php	È una funzione ausiliare per la gestione del carrello
menuL.php	È una funzione che permette di avere menu solo per gli utenti loggati
menuNL.php	È una funzione che permette di avere un menu solo per gli utenti non loggati

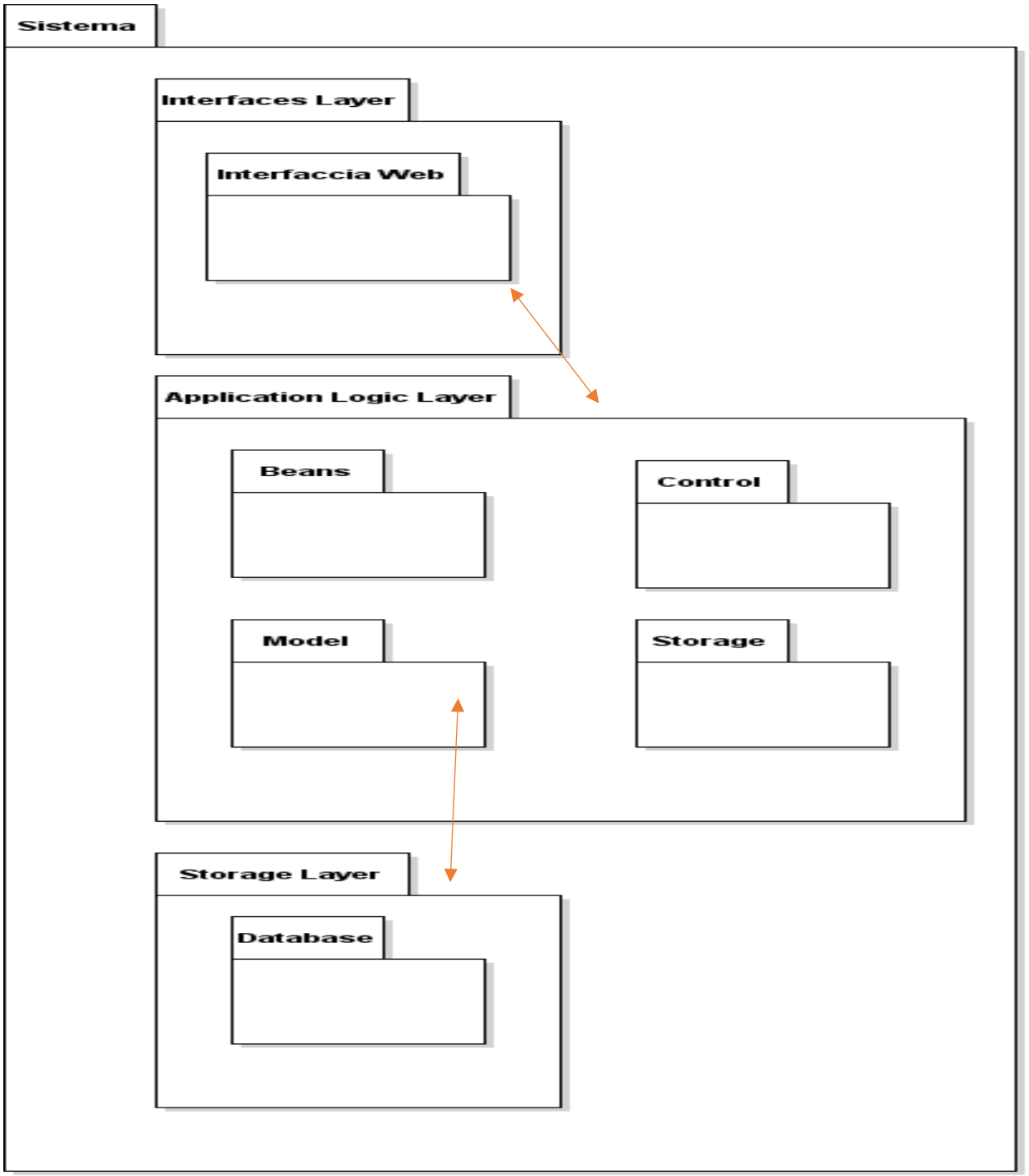
Comunicazione tra package

Il seguente diagramma mostra la comunicazione tra i vari package che compongono i layer del sistema che sono stati appena descritti. La scelta progettuale è quella di far passare ogni richiesta fatta dalle classi di interfaccia attraverso la parte di Logic Application allo storage.

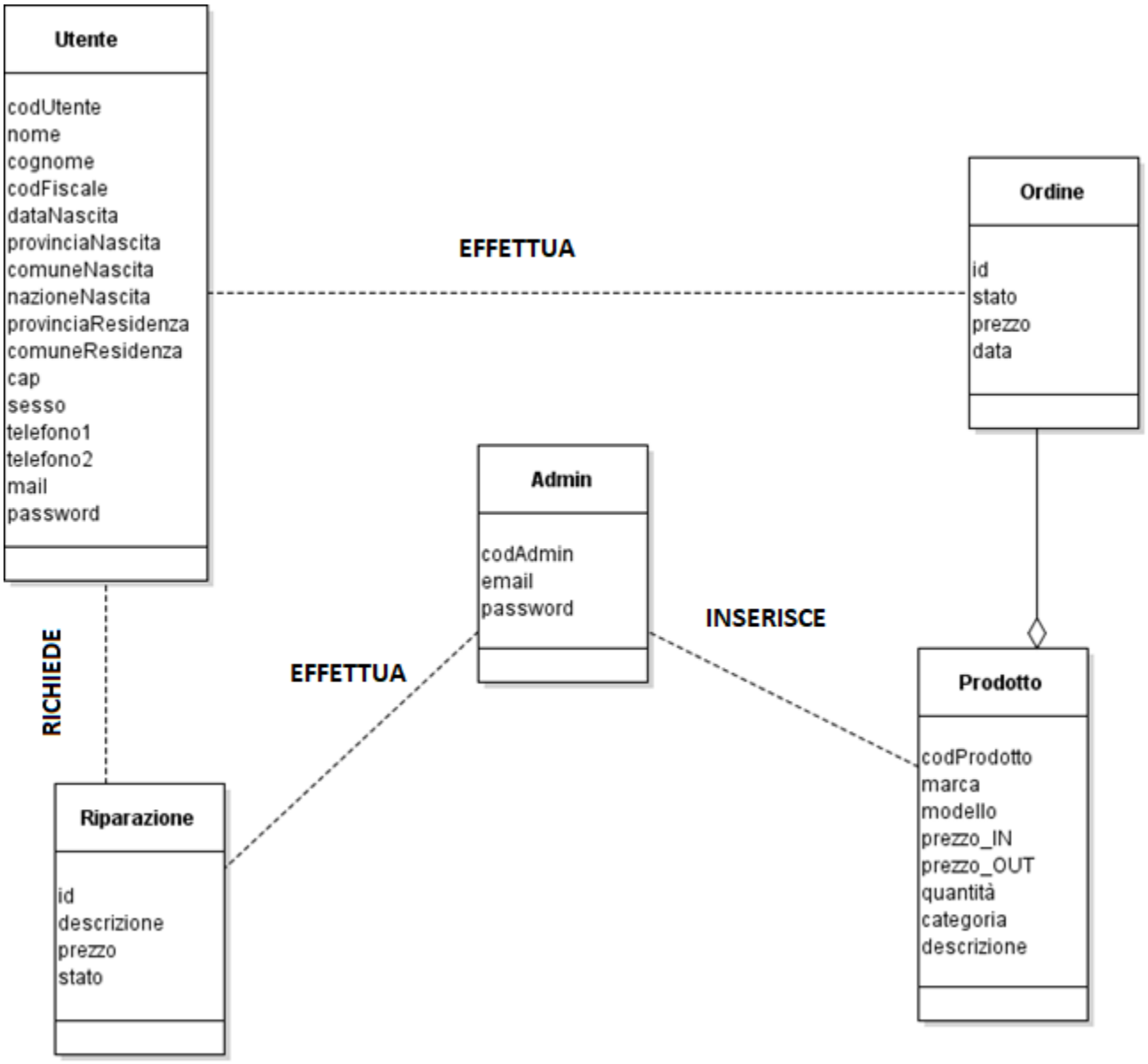
All'interno dell'Interface Layer è presente il pacchetto per la gestione della web GUI.

Il Logic Application Layer contiene i pacchetti che si occupano della logica applicativa del sistema e della sua gestione.

Infine, attraverso l'interfaccia Storage, la parte relativa al Logic Application accederà allo Storage Layer, per recuperare, aggiornare o eliminare dati persistenti dalla base dati.


Sistema
Interfaces Layer
Interfaccia Web
Application Logic Layer
Beans
Control
Model
Storage
Storage Layer
Database

Class diagram



Design Pattern

MVC Pattern

Model-view-controller è un pattern architetturale molto diffuso nello sviluppo di sistemi software.

Il pattern è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

- il model fornisce i metodi per accedere ai dati utili all'applicazione;
- il view visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;
- il controller riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti.

Questo schema implica anche la tradizionale separazione fra la logica applicativa (in questo contesto spesso chiamata "logica di business"), a carico del controller e del model, e l'interfaccia utente a carico del view.

Façade Pattern

Per la realizzazione dei servizi dei sottosistemi è stato usato il pattern Façade. Il pattern si basa sull'utilizzo di un oggetto che permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro, nonché a blocchi di codice complessi.
Nel nostro caso, ogni sottosistema, identificato nell'SDD, possiede una classe chiamata model. Questa implementa dei metodi che corrispondono ai servizi offerti dal sottosistema. tali metodi, nella loro implementazione, utilizzeranno le classi entity per eseguire il servizio da essi offerto. In output presenteranno un oggetto che servirà per la presentazione del risultato dell'operazione.

Glossario

Termine	**Descrizione**
Easy-Use	Di facile utilizzo.
Package	Collezione di classi, interfacce.